A CONCEPTUAL FRAMEWORK FOR DISTRIBUTED SOFTWARE QUALITY NETWORK

A Thesis

Submitted to the Faculty

of

Purdue University

by

Anushka H. Patil

In Partial Fulfillment of the

Requirements for the Degree

of

Master Of Science in Computer and Information Science

August 2019

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. James H. Hill, Chair

Department of Computer and Information Science

Dr. Rajeev R. Raje

Department of Computer and Information Science

Dr. Mihran Tuceryan

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

Head of the Graduate Program

Dedicated in loving memory of my Father and Uncle.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to my advisor Dr. James H. Hill, for his continuous support, motivation, immense knowledge and guidance throughout the course of my graduate study and this research. I would also like to thank my advisory committee members, Dr. Mihran Tuceryan and Dr. Rajeev R. Raje, for their insightful comments and encouragement, which incented me to widen my research from various perspectives.

I would like to thank Ms. Nicole Wittlief for providing her valuable input on my dissertation. Additionally, I am grateful to my colleagues Akshay, Siddhi, Rohit, Madhura, Snehal, and Cinthia for their support.

I am also thankful to IUPUI, and the entire staff of the Department of Computer and Information Science for providing me with the Scholarship that has allowed me to carry out my research.

Finally, I would like to thank my family: Mumma, Dada, Vahini, and Aahan for supporting me spiritually throughout my masters and my life in general.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Figure Page

ABSTRACT

Patil, Anushka H. M.S.C.I.S, Purdue University, August 2019. A Conceptual Framework for Distributed Software Quality Network. Major Professor: Dr. James H. Hill.

The advancement in technology has revolutionized the role of software in recent years. Software usage is practically found in all areas of the industry and has become a prime factor in the overall working of the companies. Simultaneously with an increase in the utilization of software, the software quality assurance parameters have become more crucial and complex. Currently the quality measurement approaches, standards, and models that are applied in the software industry are extremely divergent. Many a time the correct approach will wind up to be a combination of different concepts and techniques from different software assurance approaches [1]. Thus, having a platform that provides a single workspace for incorporating multiple software quality assurance approaches will ease the overall software quality process. In this thesis we have proposed a theoretical framework for distributed software quality assurance, which will be able to continuously monitor a source code repository; create a snapshot of the system for a given commit (both past and present); the snapshot can be used to create a multi-granular blockchain of the system and its metrics (i.e.,metadata) which we believe will let the tool developers and vendors participate continuously in assuring quality and security of systems and in the process be accessible when required while being rewarded for their services.

1. INTRODUCTION

The advent of technology is revolutionizing the widespread usage of software in all industries. Software operations play an important role in the overall working of the industries, may it be finance, healthcare, mechanical, government, public, or private sector. With an increasing rate of software utilization, advanced practices are being adopted to engineer software, and to obtain the quality metrics of the software system. Software metrics is an essential factor for the management and control of software development life cycle.

In order to ensure that an acceptable software is delivered to the user, the development team must adopt effective software testing methods to obtain the software quality metrics [2]. These metrics can be applied to each software development phase to understand software performance, quality, or the productivity and efficiency of software teams [3]. Also, these software development metrics yield the kind of insight that help to understand the type of improvement required and provide evidence of the claim or prediction [4].

The main challenge in obtaining the detailed software quality assurance metrics lays in limitations and inappropriateness of the application of existing software testing tools [5] For example, in distributed software projects where team members are geographically, temporally and culturally distant from each other, managing a project is more challenging. Moreover, it becomes even more challenging when the global teams need to work together on software metrics that drive progress towards goals and provide verifiable, consistent indicators of progress as teams are distributed all around the world and customers are not necessarily at the same site of development [3].

To address these issues organizations use different frameworks to obtain the system metrics such as, a PAMPA tool [6], where an intelligent agent tracks cost driver dominators to check the project success or failure and provides an analysis to the project

managers to modify project plans in order to reduce probability of project failure. Additionally, to compliment the advancing software engineering practises Simmons [7] introduces a software engineering expert system tool based on the PAMPA tool where the software professional can obtain metrics from CASE tool databases.

Moreover, there are very few research papers that discuss about the software assurance tools in distributed projects. In [8] Peixoto et al. discusses effort estimation in distributed development, and one of their inferences is that *"Distributed software projects are using all kinds of metrics computation techniques and none of them is being considered as adequate to be used in all cases that it has been used"*, meaning, that there is no generic software metrics computation technique or platform for distributed software projects.

In this thesis, we aim to propose a technique for distributed software quality assurance by integrating the concept of blockchain technology in the area of Software development. We have proposed a framework that creates a multi-granular and multi-dimensional view of the given system providing a platform for developers to obtain a snapshot blockchain that describes the structure of the entire system and a measurement chain that provides a detailed record of the quality assurance aspects of the components based on the tools applied.

With this understanding, the contributions of this thesis are as follows:

- First, provides a proposed framework that will help to advance software quality assurance aspects, such as acceptability, traceability, trust, and security in a software project from project initiation to project termination.

- Second, serves as a platform for distributed software team members to use different kind of software testing tools to estimate the software metrics for the project under development.

- Third, provides a platform for storing the historical record of the software development process which can assist the traceability testing and later aid in software upgrade process.

Based on our work on our theoretical framework for distributed software quality assurance, we are able to continuously monitor a source code repository; create a snapshot of the system for a given comment (both past and present); use the snapshot to create a multi-granular blockchain of the system and its metrics (i.e., metadata).

1.1 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 gives an overview of the research work Chapter 3 provides a brief overview of the blockchain technology. Chapter 4 describes the proposed approach and the associated system architectures. Chapter 5 describes the results of the empirical study with the prototype and conceptual theory. Lastly, Chapter 6 indicates the conclusion, our learning's, and what the future direction of the research.

2. LITERATURE REVIEW

This chapter discusses the research work in the area of Software Quality Assurance and analysis of different measurement tools used to estimate the software metrics. Additionally, we discuss some of the use cases of blockchain technology.

2.1 Software Quality Assurance

Simmons [6] created a PAMPA (Project Attribute Monitoring and Prediction Associate) tool to help project administrators in managing and supervising large-scale distributed projects across multiple geographically distributed systems. The PAMPA Tool is accessed using a three tier architecture, where the first tier comprises of the internet browser, second tier comprises of all the tool services as an internet server, and the third tier comprises of the knowledge base whose information is gathered from the project CASE tools supplied by any vendor. It has eighteen different dominators to validate the success and failure of the Project. The tool runs around the PAMPA Knowledge base, meaning it browses the knowledge base to reconstruct project metrics at any point. Thus, on reception of all the project information the PAMPA tool presents the project metrics at any level. Simmons [7] created the Software Engineering Expert System (SEES) tool based on the PAMPA tool. This tool was introduced to complement the evolving software engineering practices where large and advanced technologies were adopted by the distributed software teams. The SEES environment comprised of open source and open standard applications due to the availability of source code and the accumulation of application knowledge from the open source community.

Both of these tools depend solely on the knowledge base and the dominators. If the knowledge base is tampered it will entirely change the project metrics thus, providing

inaccurate information to the project administrators. However, the proposed architecture of distributed software quality network is very similar to these tools, except that the database is tamper resistance. The Immutability feature in the blockchain distributed ledger helps to maintain a permanent, indelible, and unalterable history of transactions. Therefore, in the case of our framework each data block is verified by the blockchain network and cryptographically secured by a hashing process. The blockchain linkage is unbreakable, meaning its impossible to alter or delete data after it has been validated and placed in the blockchain [9]. This architecture can thus be used to store the knowledge base and provide tamper resistance.

Fitsilis [10] presents a Social Network Analysis (SNA) model based on meta-networks where basic project entities are combined for representing communication, collaboration, and knowledge. The model defines seven major classes for modeling distributed software projects: Agents, Locations, Projects, Tasks, Knowledge, Resources and Roles. The paper proposes a framework for project analysis using SNA which they claim will enable better team selection using different measures such as the measure of knowledge used by a member team to perform a task, the power of an agent to access knowledge, resources and tasks within an organization, and the number of redundant agents per resource.

However, considering these parameters for modeling a tool for managing project is absurd as in reality the software project life cycle varies, the set of skills needed is changing, the organization used for running the projects is not fixed as everything is distributed. Using this model to monitor project and to select the team members will be unsuitable in distributed environment. Whereas in case of our framework, using blockchain technology we can establish trust between the globally distributed team members. This trust is established by the cryptographic hashing process allowing any unknown team members to work together to achieve the same goals [11]. Thus, this framework will be more preferable over the Social Network Analysis (SNA) model and will aid in managing the project and avoid unnecessary risks in the project development process.

In [12] Ibarra proposes a tool that supports the implementation of practices for quality assurance, promoting and facilitating their implementation. The paper includes a description of the tool development, as well as the validations done to the tool. The tool presented in this article was developed based on the IEEE 730 standard to cover the main activities of quality assurance. The tool proposed in this paper is made adaptable to any development methodology, provided information related to activities in each of product developments is presented. They achieved it by including five activities: First, selection of relevant products for the assurance of the quality; second, assignment of acceptance criteria; fourth, assignment of metrics to each work product according to its type and to the organizational standards; fourth, verification of configuration of the tool to identify nonconformities in its configuration before being launched; and lastly, follow-up on product measurement metrics.

The entire tool functionality is based on the pre-stated activity details however, in reality when a project is developed it goes through several remodelling . Updating and maintaining details of altered content in each and every phase will increase the project timeline and will count up more effort in software assurance process. In case of our framework, everything happens at runtime, meaning every time a commit is made the software content is added to the blockchain and tools can be implemented on these blocks to obtain the metrics moreover, the chain can be traversed back and forth to obtain different metrics at different project phases. The framework doesnt rely on any pre-stated documentation to initiate the process thus, is more suitable than the support tool.

2.2 Blockchain Technology

Following are some of the use cases of blockchain technology:

Chatterjee et al. [13]discussed the new research done in bitcoins domain using blockchain technology. They outlined the underlying concepts of blockchain. Also, they studied its applications in financial and non financial sector. The authors dis-

tinctively cited that the application of the blockchain technology in financial and non financial sector will prominently affect the issues of reliability, security and shared knowledge. They exemplified this with working of Bitcoin transactions primarily to explicate the concepts of blockchain technology focusing on the type of Proof of Work used. Lastly, the basic flaws of the system were identified and a need of exploration in the direction of minimizing the flaws and enhancing its efficiency was motivated by the authors.

Bach et al. [14] studied different consensus algorithms used in blockchain technology. They chose cryptocurrency domain for their study. Their analysis focused on algorithmic steps required by each consensus algorithm, scalability of the algorithm, the method the algorithm rewards validators for their time spent verifying blocks, and the security risks present within the algorithm. Finally, they proposed possible future trends for consensus algorithms used in blockchains.

Dabbagh et al. [15] performed bibliometric analysis of the state of the art in blockchain where Web of Science (WoS) is considered as a literature database. They collected papers from different conferences, articles from 2013 to 2018 and analyzed against five research questions. Their results revealed some valuable insights, such as number of publications and citations trends , hottest research area, top-ten influential papers, and most supportive funding bodies. They have created several graph and offered guidelines which can help both fresh and experienced research before initiating a blockchain research.

Yuan et al. [16] presented a systematic investigation of blockchain and cryptocurrencies. They discussed fundamental rationales, advantages, existing and potential ecosystems of Bitcoin. Also, they proposed six-layer reference model of the blockchain framework. In addition, they discussed different applications of blockchain and cryptocurrencies that ranges from integration of blockchain with smart contracts and IoT to using blockchain technology for the next generation of sharing economy by putting forth the idea of disintermediated models for secured immutable and P2P stored shared ledgers for the future shared economy. The authors also emphasized on the

use of blockchain technology in reducing the issues in powered freight transportation in terms of costs, efficiency and data interoperability. Their aim was to stimulate further detailed investigation and innovative research in this new direction.

Halpin et al. [17] presented ways to confront larger issues that are presented in security and privacy research for blockchain to be entirely potent in applied cryptography. Conjoining the expertise in both academia and blockchain industry the authors further explore the challenges in allowing optimum use-cases like privacy conserving file storage. This paper also deals with problems on latest cryptographic solutions deployed on Bitcoins. The solutions range from enhancing the core cryptographic primitives that is inclusive of archiving former blocks in a substantial manner and then verifying this change into newer blocks making use of hash functions or digital signature scheme. It also determines whether the various techniques proposed in the research can sustain the results presented by the use of machine-learning techniques in terms of privacy in Bitcoin. Finally, it was concluded that although there is use of fundamental security and privacy trade-off that enhances scalability and decentralization, game theory techniques could be use to delve deeper into them.

Zheng et al. [18] present a review on blockchain technology focusing on its taxonomy: public block chain, private blockchain and consortium blockchain, presents basic blockchain consensus algorithms, evaluates blockchain applications and discuss technical challenges and solutions to tackle those challenges. It also analyzes and associates the typical consensus algorithms and summarize current approaches for solving the challenges incurred. It also delves into how Artificial Intelligence can support the development of privacy in blockchain technology. The consensus algorithms have been comprehensively discussed focusing on its various types and pros and cons of each type. It also highlights the advances in consensus algorithms keeping the consensus speed as a major deciding factor while making these advancements. The paper further addresses the major challenges in the blockchain technology: Scalability, Privacy Leakage, Selfish Mining. The future aspect implementation of blockchain

technology with respect to areas like big data analytics, smart contract and artificial intelligence.

In the above research work, many authors have integrated blockchain technology in various domains whereas, our research targets the software quality assurance aspect of the software development process using the consensus theory based on the product manufacturing and audit process to advance the different aspects of software quality metrics in distributed systems.

3. BACKGROUND

This chapter provides a brief overview of the blockchain technology.

3.1 Blockchain Technology

The blockchain is the underlying technology of the Bitcoin protocol that emerged in 2008 [11]. Blockchain is a way of documenting data on the internet. Any blockchain ecosystem comprises of the following basic components:

- **Block**: The data is documented in the form of a growing list of records called as blocks.

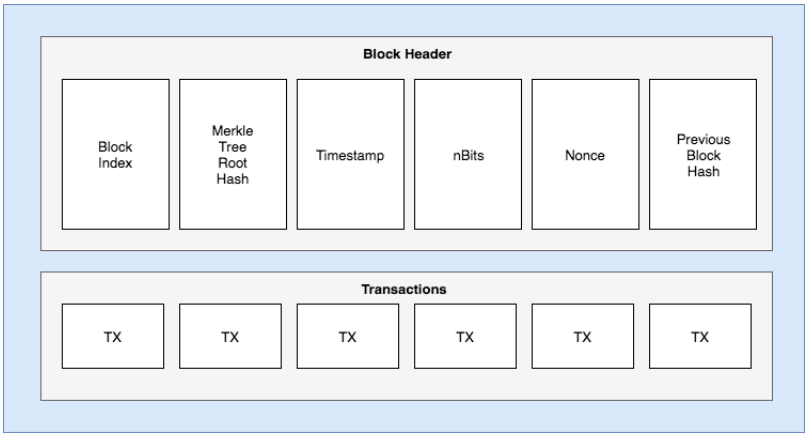

Fig. 3.1. Block Structure

As shown in Fig. 3.1 a block comprises of two components: block header and block body [19]. The block header consists of following elements:

  - Block index: It is an internal unique number for the block.
  - Merkle tree root hash: A merkle tree is basically a tree whose leaf nodes are labelled with the hash of the transaction in the block. Merkle tree root

hash is the hash of the hashes of transactions in the block thus, represented as the hash of the block.

- Timestamp: It is the timestamp when the block is created.
- nBits: It is the target threshold of a valid block hash.
- Nonce: It is an arbitrary number of byte field which starts with the 0's and increases for every hash computation.
- Previous block hash: It is the hash of the previous block which helps to link the blocks.

The block body comprises of all the transactions in that block. The capacity of transactions per block varies depending on the size of the block and the size of each transaction [19].

A genesis block is the first block (Block 0) of a Blockchain. It is common ancestral parent of all the new blocks created. At any time, if we traverse backward we will eventually reach the genesis block i.e. block 0 at the end [20]. The structure of the genesis block is priorly encoded within the blockchain framework and cannot be tampered.

Furthermore, it combines the openness of the internet with the security of cryptography thus, providing a safer way to verify the information and establish trust. Additionally, it eliminates the centralized system by using the distributed system with distributed ledger. There is no central authority supervising anything in the blockchain while there is a consensus mechanism by which this decentralized network comes to a consensus on certain matters (will be explained in details in the next subsection). The distributed ledger database is spread across different locations and nodes on a peer-to-peer network, where each replicates and saves an identical copy of the ledger and updates itself independently.

- **Cryptography**: All the information on this ledger is securely and accurately stored using cryptography and can be accessed using keys and cryptographic

signatures. The key to a blockchains security is the cryptography algorithm: Hash. Hash is nothing but a unique string of characters generated from a string of text using a mathematical function. A cryptographic hash function is a specific class of hash functions which has different properties making it optimal for cryptography [21]. Every time the information is supposed to be stored, a hash function is used to create a hash of the information. This hash is used to secure the block.

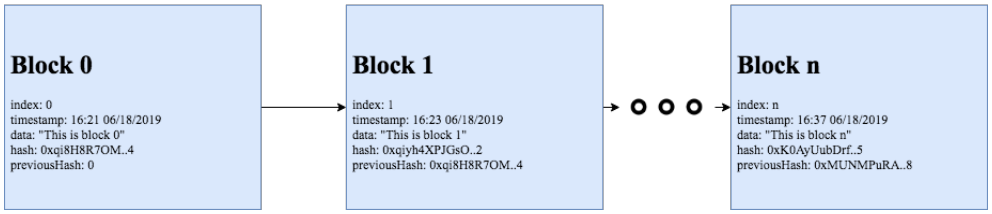

Fig. 3.2. Blockchain Architecture

As shown in Fig. 3.2 The hash of one block is added to the next block, in this manner a linkage is formed between the blocks and the same process is continued throughout the chain. If there is any attempt to alter a previously created block, the hash that's encoded in the next block won't match up anymore. Thus, once the information is stored it becomes an immutable database. Blockchain is designed to store information in a way that makes it virtually impossible to add, remove or change data without being detected by other users.

- **Consensus Theory**: When a ledger update happens, each node constructs the new transaction , and then the nodes vote by consensus theory on which copy is correct [19]. Consensus theory is nothing but a protocol that allows a blockchain to be updated by validating that each and every block in the blockchain is true. Following are some of the common consensus algorithms in blockchain

  - Byzantine fault tolerance: Byzantine faults was identified and characterized by Leslie Lamport as the Byzantine Generals Problem [22]. Byzantine

fault causes nodes to behave erratically. The Byzantine faults are the most severe and difficult to deal with. In this algorithm, the network is Byzantine fault tolerant as long as the number of faulty nodes do not exceed one third of all the nodes in the network. If the faulty nodes are not detected, the nodes will be able to transmit and add false transaction to the blockchain. Thus, the Byzantine fault tolerance is necessary for blockchain validation.

- Proof of work: In proof of work, each node of the network is computing a hash value. Every time a hash value is computed, the nonce is incremented to get a value equal to or smaller than a certain given value. When one node reaches the desired value, it would broadcast the block to other nodes in the network. After reception of the block, all other nodes must mutually validate the rightness of the hash value. If the block is validated, other miners would append this new block to the blockchain [19].
- Proof of stake: Proof of stake is an alternative to Proof of Work. In case of proof of work, all the miners have to do a lot of computation which leds to wastage of too much resources where as, in case of proof of stack the miners have to prove the ownership of the amount of assets. As it is believed that people with more assets are likely to attack the network [19]. In this case, the miners are selected based on the asset balance which is quite unfair as the miner with highest asset will dominate the network. Proof of Stake works well in cases where resource exploitation is expensive.

These algorithms make sure that the miners are not faulty and the work contribution by the miners is valid. Once a consensus has been determined, all the other nodes update themselves with the new, correct copy of the ledger.

Following are the some of the features of Blockchain Technology:

- **Decentralized**: One of the most exciting aspects of blockchain technology is that it is entirely decentralized, where nodes are only connected to peers. This

removes the need for powerful central authorities and instead hands control back to the individual nodes in the peer to peer network. A lack of a central authority makes the system fairer and more secure. In context of software development lifecycle this will help us to avoid single point failures as the entire process will be decentralized. The decentralized systems are less likely to fail accidentally as they rely on many distinct components. Furthermore, as most of the components are distributed there is no single point of failure, additionally theres no one point of attack that would disarm the entire system which makes it attack resistant. Each of these aspects scale with the level of decentralization in the system. The more decentralized the network, the more fault tolerant, and attack resistant it becomes.

- **Distributed Ledger**: A distributed ledger is a database that exist across several nodes or locations. It provides a verifiable and auditable history of all information stored on that particular dataset. All content stored in the distributed ledger are time stamped and given a unique cryptographic signature. All of the nodes on the distributed ledger can view all of the records. All of these records are immutable in nature - this property offers a way to securely and efficiently create a tamper-proof log of all the activities.

- **Immutability**: Immutability is the most important benefit a blockchain provides. The blocks are cryptographically linked to each other, tampering this cryptographical linkage requires humongous computation power which involves expensive cost. Moreover, in practise recording huge data involves huge storage expenses so we record the hash of the data in the blocks which offers tamper resistance. Whenever the input of a file is changed, its corresponding hash value will also change. Regardless of where you store your document, whether in a centralized system or in a distributed database, you can still verify the document has not been tampered with by computing its hash and comparing it to the stored hash.

4. PROPOSED APPROACH

In this chapter, our proposed approach for distributed software quality network is discussed. The chapter also describes the framework, blockchains, and the consensus theory. A prototypical system is designed, implemented, and studied.

4.1 Distributed Software Quality Network Architecture

In this section, we discuss the system architecture for Distributed Software Quality Network (DSQN). The main goal is to create a multi-granular and multi-dimensional view of the given system.

The system architecture consists of following components:

- DSQN Framework
- Database
- User Interactions

We discuss each of them in detail below.

4.1.1 DSQN Framework

The DSQN Framework is developed using the Blueprint Framework [23]. The framework acts as a web-based API component of the system. The Blueprint Framework has two main foundational building blocks: Routers and Controllers. The framework execution logic is provided by the controller. The controller handles the request input, processes it, and then sends a response to the request. Routers on the other hand are client facing interfaces where all the request are made. This approach is used to add and fetch data, and contains GET, POST, and PUT type functions. The

framework interacts with the database by routing URLs to the databases to store and retrieve data requested by the clients.

4.1.2 Database

The database is designed using Mongoose (object document modeling (ODM)) layer with MongoDB [24] [25]. We have defined schema for our data models, so the documents follow a specific structure with predefined data types. These documents are stored in the form of JSON style which provides us an easy parsing and faster execution of the data. Furthermore, the database index makes it possible to quickly retrieve data. Additionally, Mongoose has built-in validation for schema definitions which saved our time from writing a bunch of validation codes. Moreover, the optional pre and post save operations made it easier to define hooks and custom functionalities.

The framework comprises of following models:

- Component: This model is for creating a schema inheritance for the different types of components: System, File, and Unit. This schema enables us to have different models for the above components. This model helps us to store all the components under same collection. As shown in Fig 4.1 The component schema comprises of a discriminator key, which in our case is the *type*. The *type* attribute helps to discriminate different types of components: system, file, and unit. The component model design schema comprises of following properties:
  - *name* : The name of the component, SchemaType - String
  - *hash* : The hash of the component computed using crypto-js/sha256 nodejs library [26], SchemaType - String
  - *content* : The entire content of the component, SchemaType - String
  - *version* : The version of the component, SchemaType - Number

  Fig 4.1 describes the different types of components schemas using the discriminator key *type* defined in the component schema. The system schema used to

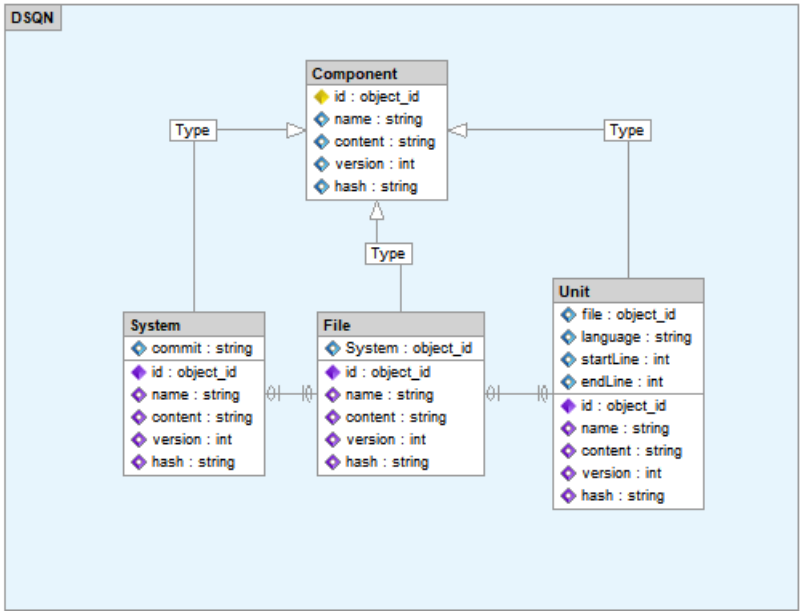

Fig. 4.1. ER Diagram: Component

describe the system component. The System model design schema comprises of following property:

– *commit* : The commit hash value, SchemaType - String.

The File schema describes the file component. The file model design schema comprises of following property:

– *system* : The reference to the system component which contains this file, SchemaType - ObjectId.

The unit schema describes the unit component. The unit model design schema comprises of following property:

– *file* : The reference to the file component which contains this file, SchemaType - ObjectId.

– *language* : This property states the programming language computed using the language-classifier npm package [27], SchemaType - String.

  - *startLine* : The start line of the unit content being extracted, SchemaType - Number.
  - *endLine* : The end line of the unit content being extracted, SchemaType - Number.

- Snapshot: This model is for storing the block content for the snapshot chain. As shown in the Fig 4.2 The snapshot model design schema comprises of following properties:
  - *component* : The reference to the component, SchemaType - ObjectId.
  - *hash* : The hash value of the current snapshot block, The SchemaType - String.
  - *prev* : The hash value of the previous snapshot block, The SchemaType - String.

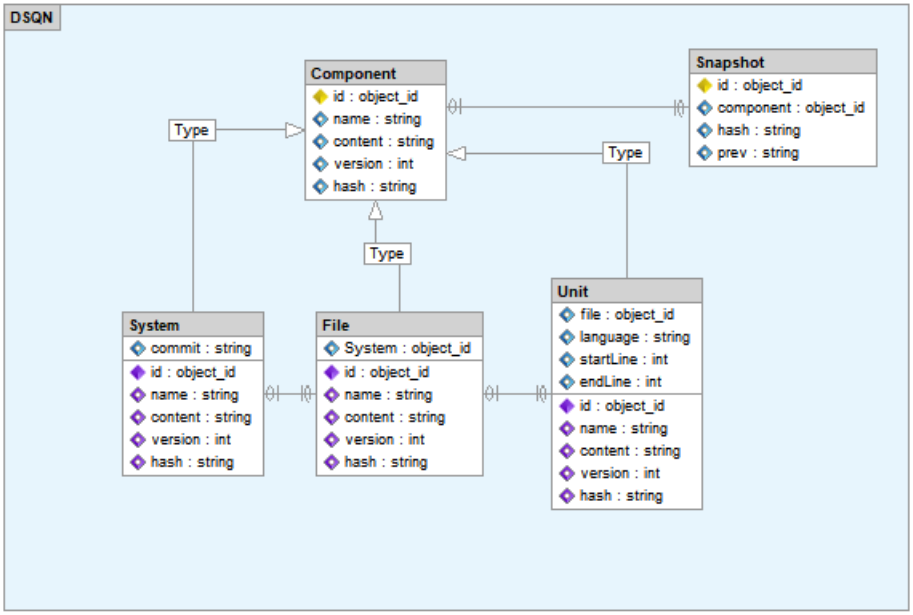

Fig. 4.2. ER Diagram: Snapshot

- Tool: This model is for storing the information of the tool. It comprises of a PropertyDesc schema as shown in the Fig 4.3. The PropertyDesc schema stores

all the description about the metrics evaluated by the tool which is further referenced in the measurement schema for the ease of data handling. The tool model design schema comprises of following properties:

- *name* : The name of the tool, SchemaType - String.
- *properties* : The array of different property metrics evaluated by the tool. It references the PropertyDesc schema, SchemaType - ObjectId.

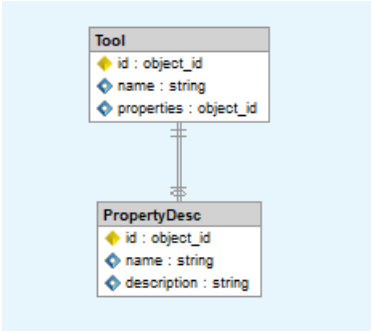

Fig. 4.3. ER Diagram: Tool

- Worker: This model is for storing the information about the workers. The workers are categorised into two types: miner and validator. The miner workers are used for the mining process while the validator workers are used for the validation process.

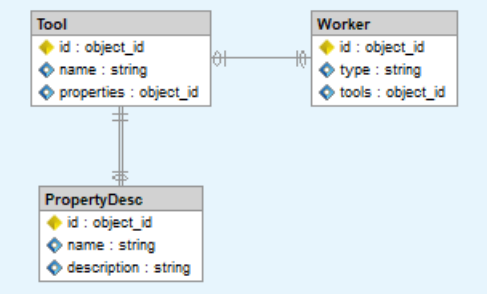

Fig. 4.4. ER Diagram: Worker

As shown in the Fig 4.4 The worker model design schema comprises of following properties:

- *type* : The type of the worker, SchemaType - String.
- *tools* : The array of the tools set up by the workers in their environment. It references the tool schema, SchemaType - ObjectId.

- Toolpool: This model is for storing the tool request. As shown in the Fig 4.5. The toolpool model design schema comprises of following properties:
  - *component* : The reference to the component, SchemaType - ObjectId.
  - *tool* : The reference to the tool, SchemaType - ObjectId.
  - *miners* : The array of miners who have the tool in their environment. It references the miners: worker schema, SchemaType - ObjectId.

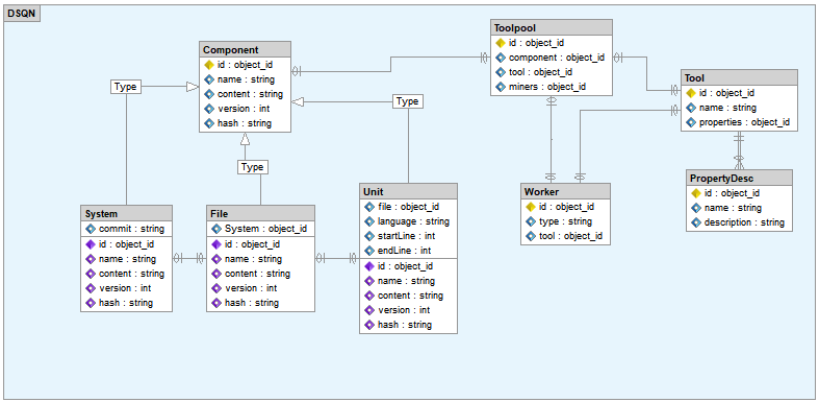

Fig. 4.5. ER Diagram: Toolpool

- Mining: This model is for storing the information about the mining process. As shown in the Fig 4.6 The mining model design schema comprises of the following properties:
  - *miner* : The reference to the miner workers, SchemaType - ObjectId.

- *start-time* : The start time of the mining process, this value is later used as random for the mining process, SchemaType- Date.
- *end-time* : The end time of the mining process, this value is later used as nonce for the mining process, SchemaType- Date.
- *toolpool* : The reference to the toolpool i.e. the tool request, SchemaType - ObjectId.

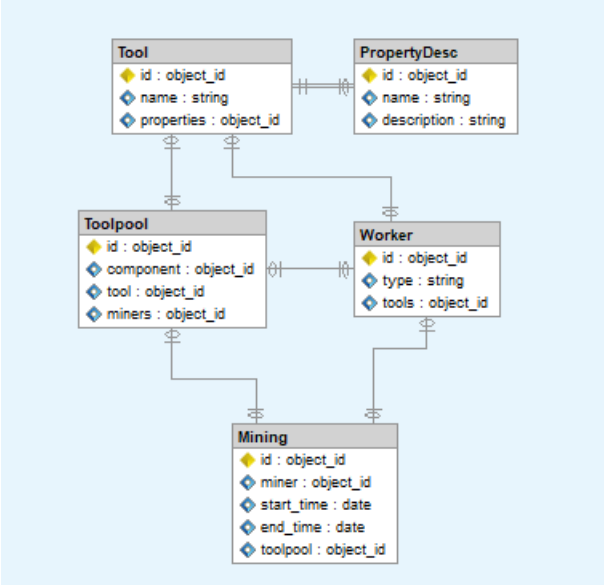

Fig. 4.6. ER Diagram: Mining

- Measurement: This model is for storing the block content for the measurement chain. As shown in the Fig 4.7 The measurement model design schema comprises of the Measure schema which is used to store all the metrics information consisting of following properties:
  - *property* : The reference to the tool property information, SchemaType - ObjectId.
  - *value* : The metric value obtained after implementing the tool over the component, The SchemaType here is mixed to accept values of any type.

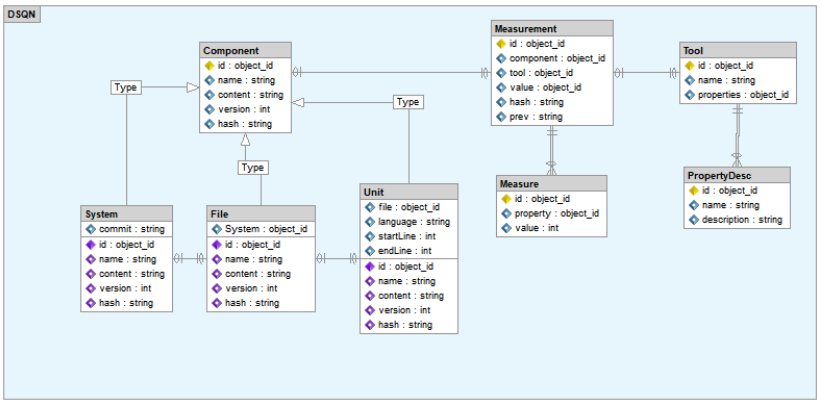

Fig. 4.7. ER Diagram: Measurement

Altogether, the measurement schema comprises of following properties:

- *component* : The reference to the component, SchemaType - ObjectId.
- *tool* : The reference to the tool, SchemaType - ObjectId.
- *value* : The array of all the metrics obtained after implementing the tool. The elements in the array reference to the measure schema, The SchemaType - ObjectId.
- *hash* : The hash value of the metrics block, The SchemaType - String.

4.1.3 User Interactions

Following are several interactions of the end users with the framework:

- Client Request

  In this interaction, end users access the services provided by the framework. They send the requests to the framework to execute different services. The client pays for the services.

- Mining Process

  In this interaction, end users play the role of a miner. They pick the request

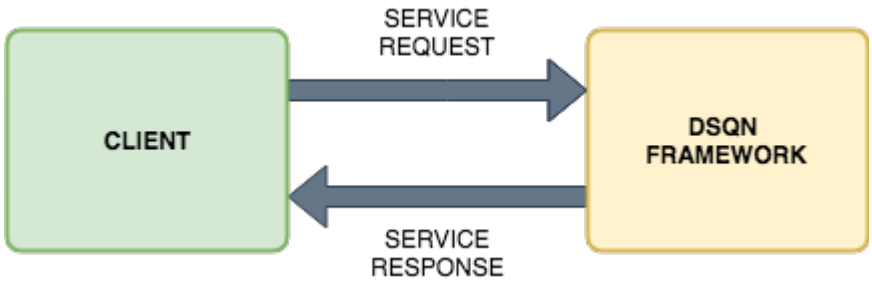

Fig. 4.8. User Interactions: Client Request

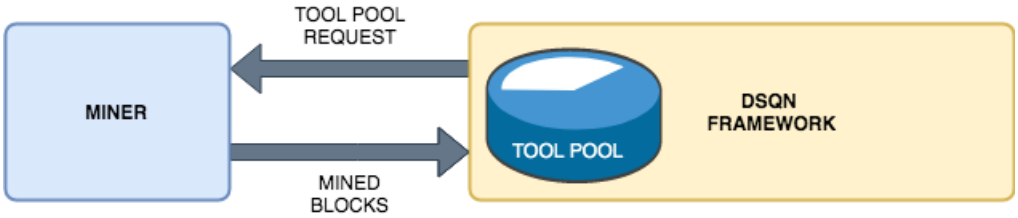

Fig. 4.9. User Interactions: Mining Process

from tool pool and start with the mining process. Miners are rewarded for the tool mining efforts.

- Validation Process

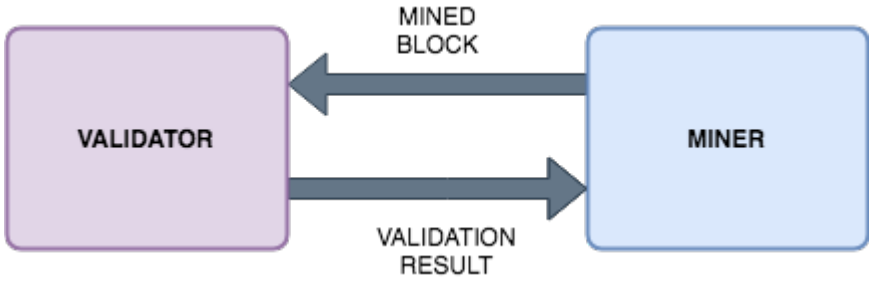

Fig. 4.10. User Interactions: Validation Process

In this interaction, end users play the role of a validator. They receive the mined blocked from the miners and validate it. Validator is rewarded for the validation process.

4.1.4 Blockchain

In this section, we discuss the types of blockchains created by the framework. For a given project, the framework creates two types of blockchains as follows:

- Snapshot Blockchain

  The snapshot blockchain is a chain of snapshots at component-level. The entire project is divided into three components:

  - System: It comprises of collection of all the files
  - Files: It comprises of the individual files
  - Units: It comprises of the different units (methods/functions/modules) in each file.

  The structure of the snapshot block is defined as:

  Snapshot {*Component id, Hash, Previous Hash*}

  Where, Component id is the components reference id and Hash is the hash of current block, and Previous hash is the previous blocks hash.

  The snapshot blockchain describes the structure of the entire system. In distributed development environment, where team members are distinct from each other, this chain can serve as a secure and tamper resistant source for maintaining a record of the entire software development process. This chain can be traversed backwards and forwards to trace the software development process. Thus, we believe that snapshot chain serves as a good way for traceability testing of the system while the project is still under development.In the software upgrade process, the snapshot chain will serve as a great source for the record of uneditable history. These records can be used to research the work history, get more insights of the project and track records.

- Measurement Blockchain

A chain of measurements at component-level based on the set of tools applied.

The structure of the measurement block is defined as:

Measurement {*Component id, Tool id, Measure, Hash*}

Where, Component id is the components reference id, tool id is the tools reference id, measure is an array of software metrics and Hash of the block comprises of component hash and previous blocks hash.

The measurement chain provides a detailed record of the quality assurance aspects of the components based on the tools applied. This chain serves as an excellent way to track the project and measure the performance while the project is still under development. The Software Project Managers can use this secure and tamper resistant chain as a resource to anticipate problems. Thus, the chain serves as an adequate platform for early discovery and correction of technical and management problems that can be more difficult or costly to resolve later. Moreover, these metrics help understand capabilities of the software and improves the predictability of the system.

4.2 Consensus Theory

In this section, we discuss the consensus algorithm for the blockchain mining and validation process. The consensus theory is based on the product manufacture and audit process. Like any product manufacturing company whose process depends on the worker and auditor our consensus theory depends on the miners and validator.

The consensus theory based on the following components:

- Tool pool: Tool pool is a collection of all the client request. Each element (request) in the tool pool contains the id of the component whose metrics are supposed to be computed and id of the tool which is supposed to be used to compute the quality metrics.

- Miners: In the manufacturing company there are several workers working on different operations based on their proficiencies, in our case we have different miners having different tools setup for computing the metrics.

- Validator: In the product manufacturing process, there is a validation process to check the results of the operations being perform and each step of a manufacturing process is controlled to assure that the finished product meets all design characteristics and quality attributes including specifications. In similar fashion, we have validators who validate the results obtained from the miners and assure that valid software quality metrics are obtained. To avoid the faulty validator issue we have used the byzantine theory [22] to decide the number of validators for validating the content.

The consensus algorithm works as follows:

- The client selects the tool and component and sends a request to the framework. The request contains the tool id and component id for references.

- The framework on reception of this request add the request in the tool pool. When the request is added into the tool pool, a notification is sent to the miners having that tool implemented in their environment.

- The miners then pick the request from the tool pool and start executing the request. Every miner takes different computation time and requires different computational power to execute the request. The miner who completes the task first encrypts their data using their private key and the public key is distributed to the validators. This encrypted data is forwarded to the validator for validation.

- The validators on reception of the result decrypt the data using the public key. This encryption mechanism will prevent third-parties from accessing the block content while it's transferred from miner to validator. The validation process is

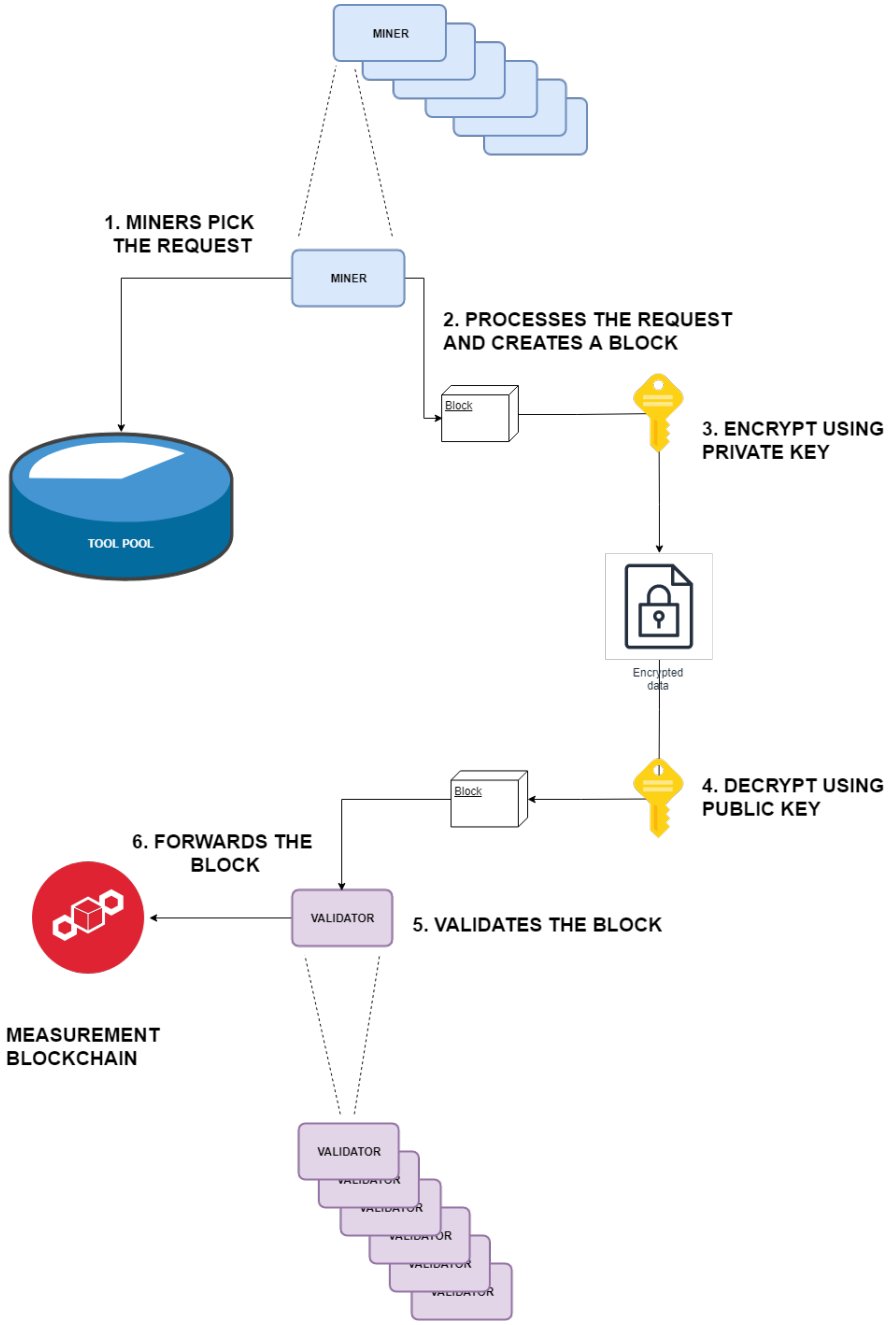

Fig. 4.11. Consensus Theory

conducted by a set of validators based on the byzantine theory [22]. Once the validators reach consensus the block is added into the chain .

4.3 System Workflow

In this section, we discuss the system workflow. The system workflow involves two phases:

4.3.1 Phase 1: Snapshot Blockchain

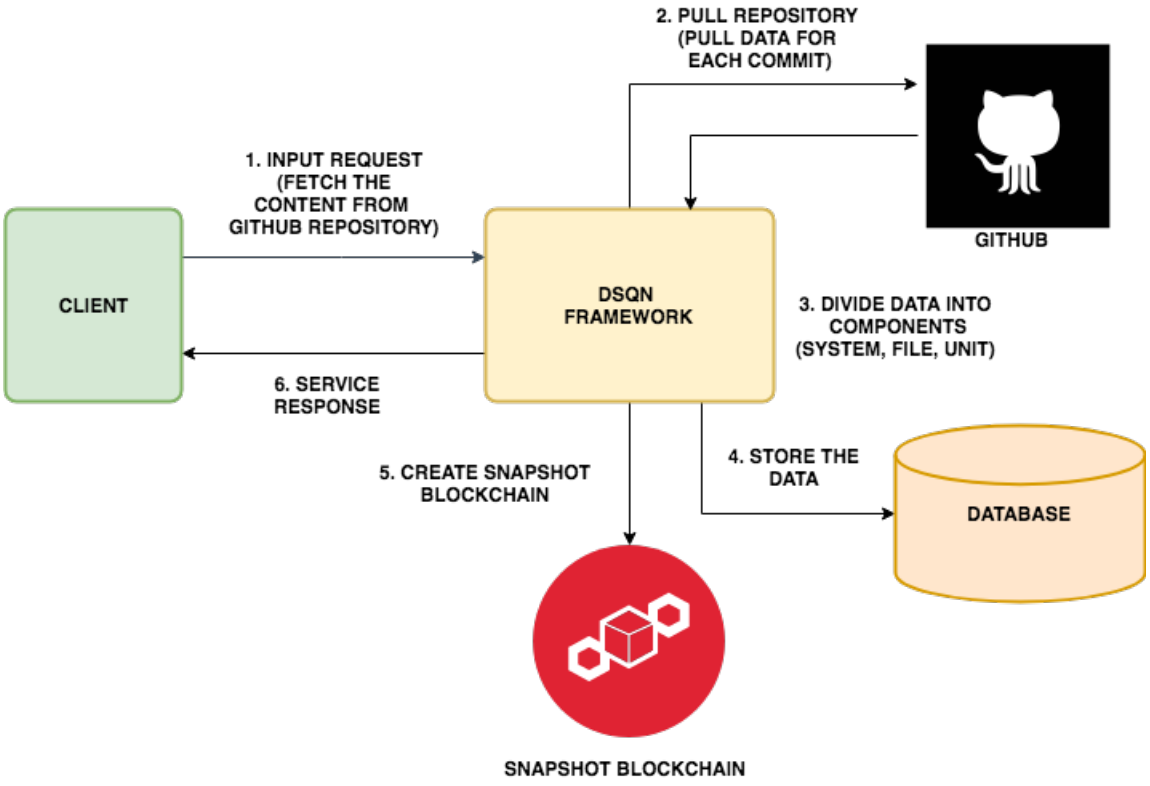

Fig. 4.12. Phase 1: Snapshot Blockchain

In this phase, a blockchain is created which describes the entire structure of the system. It involves several snapshots of the components being added while the developing the system.

- The user sends an input request to the framework to fetch content from the GitHub repository and to add it to the database. The input request contains the details of the GitHub repository.

- The framework on reception of this input request, extracts contents for each commit. The data obtained from each commit is divided into three components: System, Files, and Units. These components are then stored in the database, each component has its own specific structure with predefined data types.
- After each component is added into the database, a block is created for each of the added component to create the snapshot blockchain. This blockchain describes the structure of the entire system.

4.3.2 Phase 2: Measurement Blockchain

In this phase, a blockchain is created which describes the quality aspects of the system. It involves implementing several tools on the components and obtaining quality metrics for same.

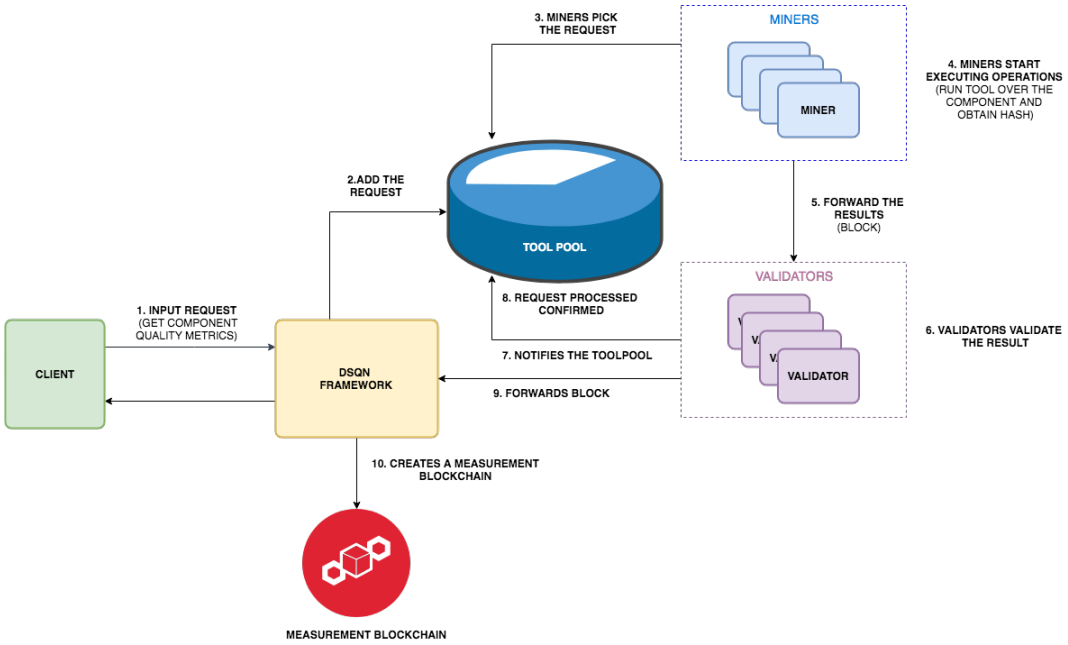

Fig. 4.13. Phase 2: Measurement Blockchain

- The framework serves as a platform for the developers to run their system through different tools and obtain the quality metrics.

- The user selects the tool and component whose quality metrics are supposed to be computed.
- The framework then adds these tool and components in the tool pool. The tool miners pick the tool pool request and run the tool over that component. The results are then forwarded to the validators. The validators validate the results.
- If the results are successfully validated a block is created and added to the measurement blockchain else, the miner is red flagged, and the validators wait for other miner to forward the result. This measurement blockchain describes the quality aspects of the entire system.

5. EXPERIMENTATION, ANALYSIS, AND DISCUSSION

This chapter discusses multiple illustrations that were carried out in order to empirically validate the proposed theory for creating a distributed software quality network.

5.1 Experimental Setup

The distributed software quality network framework comprises of two main elements: Application Server and Database. Following is the detailed information about the experimental setup for the above elements.

5.1.1 Application Server

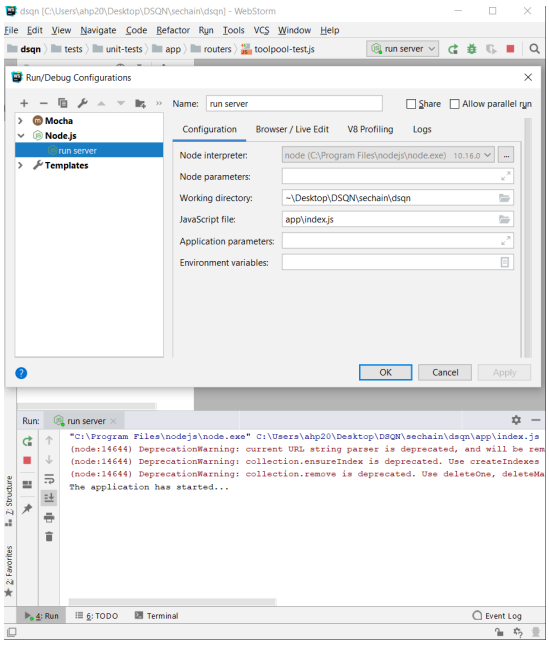

Fig. 5.1. Application Server

For the purpose of experimenting, we created a application server that was running in a local server configuration root folder using Jet Brains WebStorm IDE [28] as shown in Fig 5.1 We had the URL address *"http://localhost:5000"* to access the application sever.

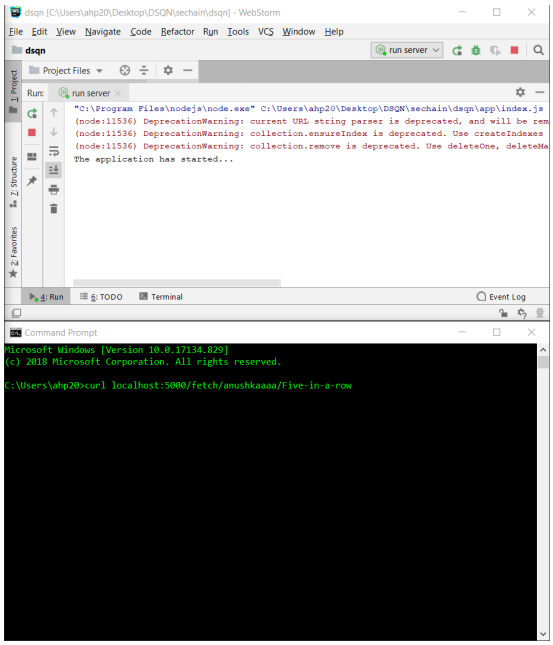

Fig. 5.2. Command Line Interface

As shown in Fig 5.2, we used the command line interface (CLI) for sending all the URL request to the application server and for receiving the response from the application server.

5.1.2 Database

In order to obtain the data required for this experiment, we decided to fetch data from *GitHub* repository. As GitHub repository is a development platform for the developers to manage their projects, it served as an excellent source to fetch the different phases of the development process of a project [29]. To clone and obtain the files from the GitHub repository, we have used the *NodeGit* library in our framework.

As, NodeGit can be quickly and painlessly installed via NPM it provided an ease to clone the repositories and fetch the content [30]. We made use of Robo 3T GUI for viewing the MongoDB records [31] [25]. For the purpose of experimenting, we chose different public repositories ranging from 3 commits to 20,000 commits.

5.2 Experiment 1: Creating a Snapshot Chain

The goal of this experiment was to create a snapshot chain for the different phases of the project development cycle. To obtain different versions of the project phase, we fetched data for each commit made in the project repository.

5.2.1 Setup & Results

We setup the experiment as discussed in Section 5.1. The files were fetched from the project repository in the following sequence:

- Cloned a repository into a folder named repositories
- Traversed branch by branch, starting from the Master branch
- Obtained the history of the commits and pulled content for each commit.

As described above, the content was pulled commit by commit. For each commit the entries were recorded, then the fetched content was divided into three components: System, File, and Unit. A block was created for each of these components and were eventually added to the Snapshot blockchain. Fig 5.3 shows the system model which comprises of following attributes: *type, hash, name, commit, content, and version.* Fig 5.4 shows the file model which comprises of following attributes: *type, hash, name, system, content, and version.* Fig 5.5 shows the unit model which comprises of following attributes: *type, hash, name, file, content, language, start line, end line, and version.* We were thus able to successfully obtain a snapshot chain for the given repository. We ran the experiment for different sets of commits ranging from 3 commits to 20,000 commits. The results are shown in the Fig 5.6

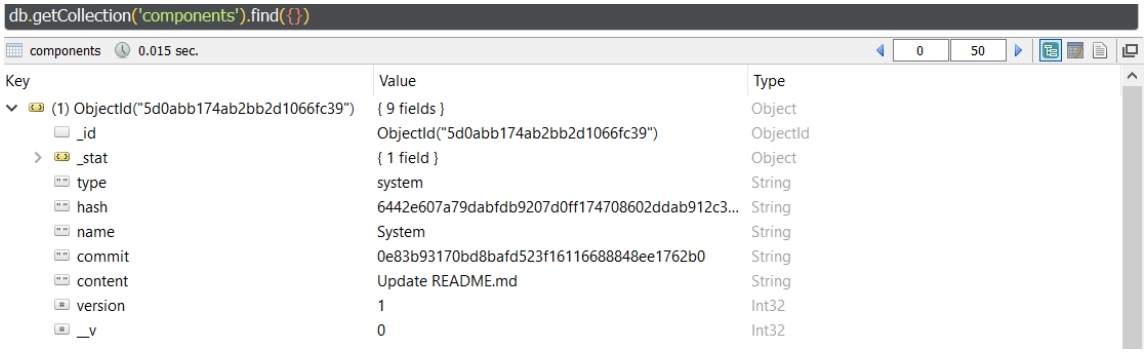

Fig. 5.3. System Model Schema

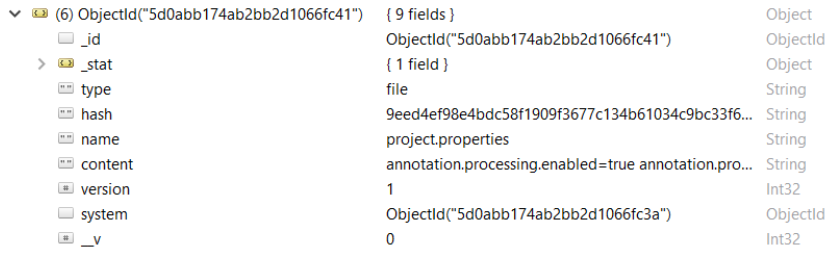

Fig. 5.4. File Model Schema

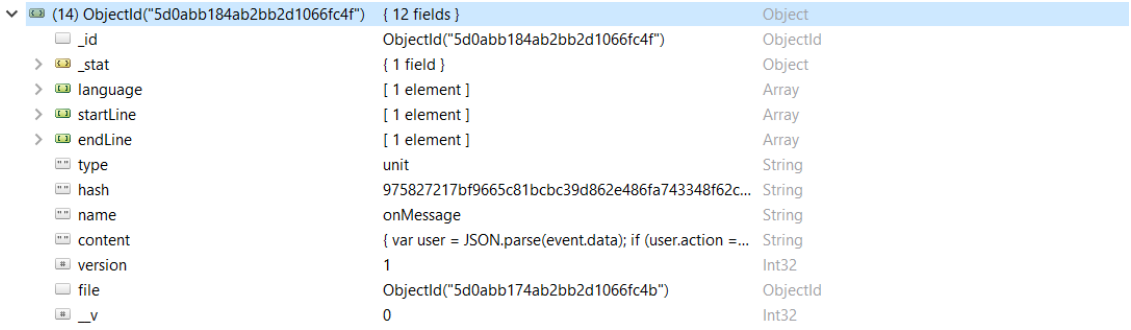

Fig. 5.5. Unit Model Schema

5.2.2 Analysis

From the results, we observed that the framework was able to successfully pull all the files and create a snapshot block for the entire project including the following components: System, Files, and Units. The information was accurately stored

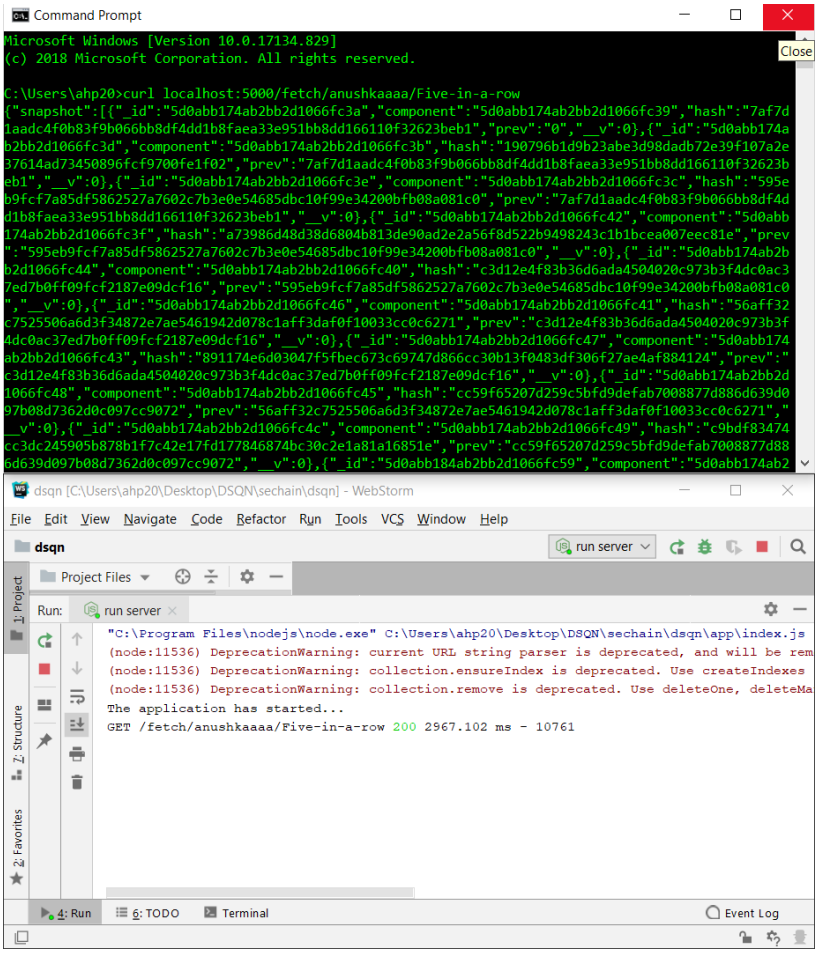

Fig. 5.6. Snapshot Chain

in the block, meaning that the data was stored as specified in the schema. Additionally, we observed the cryptography linkage between the adjacent blocks. As shown in Fig 5.7, the hash of the block with *ObjectId("5d0abb174ab2bb2d1066fc3a")* is *7af7d1aadc4f0b83f9b066bb8df4dd1b8faea33e951bb8dd166110f32623beb1* and the succeeding block with *ObjectId("5d0abb174ab2bb2d1066fc3d")* has its current hash value as *190796b1d9b23abe3d98dadb72e39f107a2e37614ad73450896fcf9700fe1f02* and previous hash as *7af7d1aadc4f0b83f9b066bb8df4dd1b8faea33e951bb8dd166110f32623beb1*. This Hash value link each block to its predecessor, by holding a hash value of the data in the previous block.

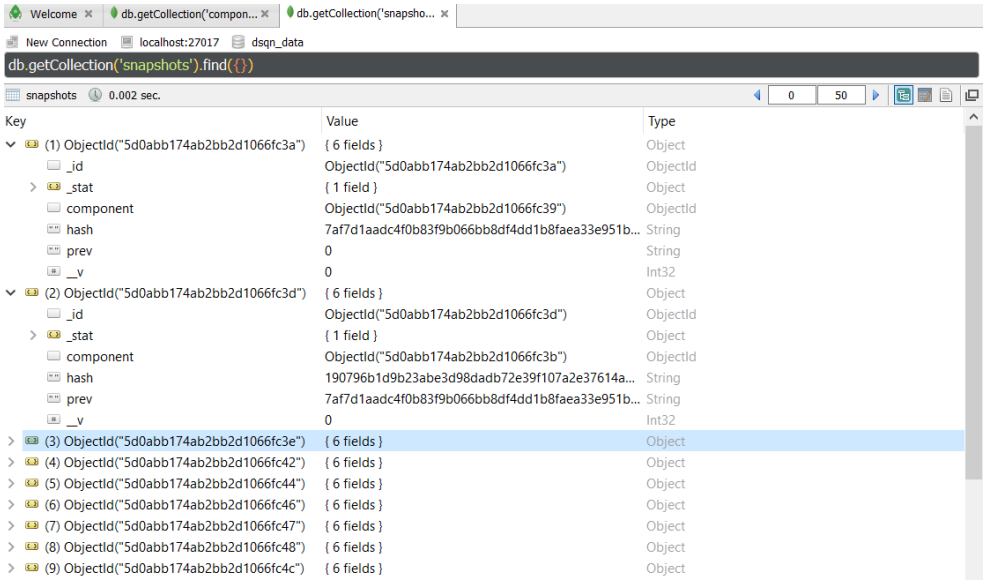

Fig. 5.7. Snapshot Chain

The overall time taken by the framework to create a snapshot chain for different repositories ranging from 3 commits to 20,000 commits and branches ranging from 1 branch to 5 branches is displayed in the Graph 5.8 A few deviations can be noticed in the Graph 5.8 The variability in the number and size of files committed per repository result in these deviations. Moreover, the time taken for each file per commit also varies. To analyse in more depth, we took a snapshot of the processing time for one of these repositories considering initial 64 commits in the Graph 5.9 Here, several files are committed per commit. The size of each file varies and correspondingly the processing time per file varies. From graph 5.8 we can notice that as the number of commits increase the processing time increases. Although, after analyzing graph 5.9 we inferred that the overall processing time depends on several parameters such as number of commits, number of branches, and number and size of files committed per commit.

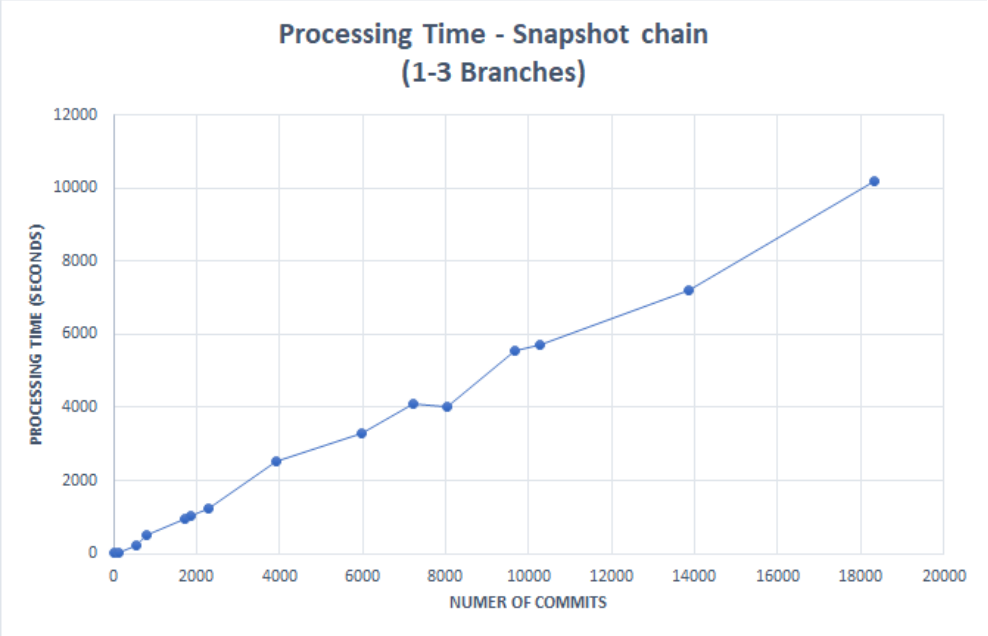

Fig. 5.8. Processing Time - Snapshot chain (1-5 Branches)

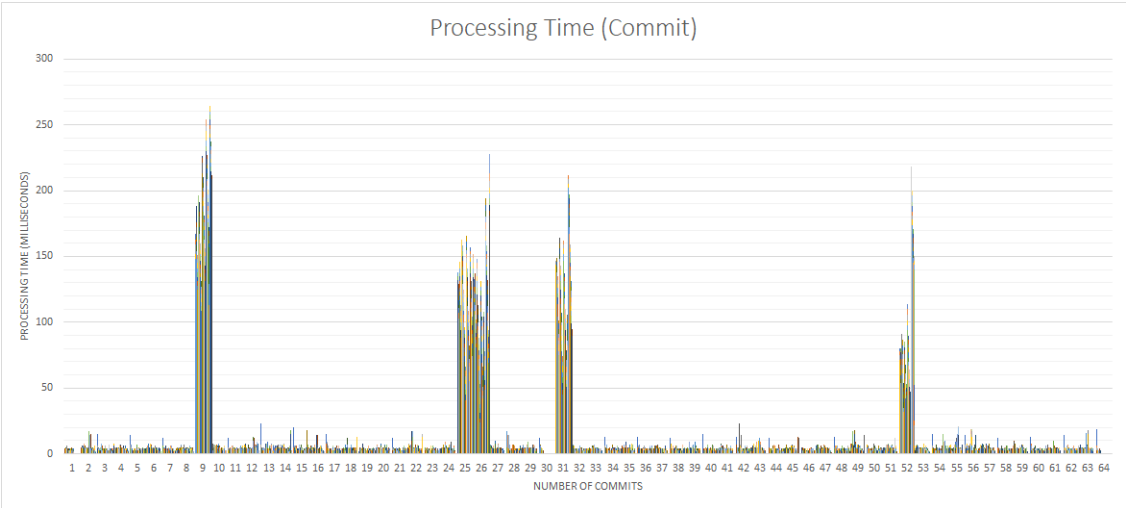

Fig. 5.9. Processing Time for files per commit

5.3 Experiment 2: Creating a Measurement Chain

The goal of this experiment was to create a measurement chain by applying tools on the different components of the project.

5.3.1 Setup & Results

We setup the experiment as discussed in Section 5.1. For the purpose of experimenting, we considered fetching the JavaScript repositories and applied the JavaScript software measurement tool. The tool is setup in the environment where the application server is running. We have considered the escomplex tool, it is a npm package for Software complexity analysis for JavaScript projects. It reports following metrics: Lines of code, Number of paramters, Cyclomatic complexity, Cyclomatic complexity density, Halstead metrics, Maintainability index, Dependencies, First-order density, change cost, and core size [32]. The data related to the tools is stored as shown in the Fig 5.10 We applied the tools on different components i.e. Systems, Files, and Units and obtained the measurements for each of these components. The measurements obtained are stored in the database as shown in the Fig5.11

5.3.2 Analysis

From the results, we observed that the framework was able to successfully apply tools on each of these components: System, Files, and Units. The information was accurately stored in the database, meaning that the data was stored as specified in the schema. Furthermore, we analyzed that tools can be applied at any stage of the software development, meaning in our experiment we applied tool on components obtained from initial commit to the latest commit. In this manner, the developers can apply tools on the same component for different commits and analyze the software advancement.

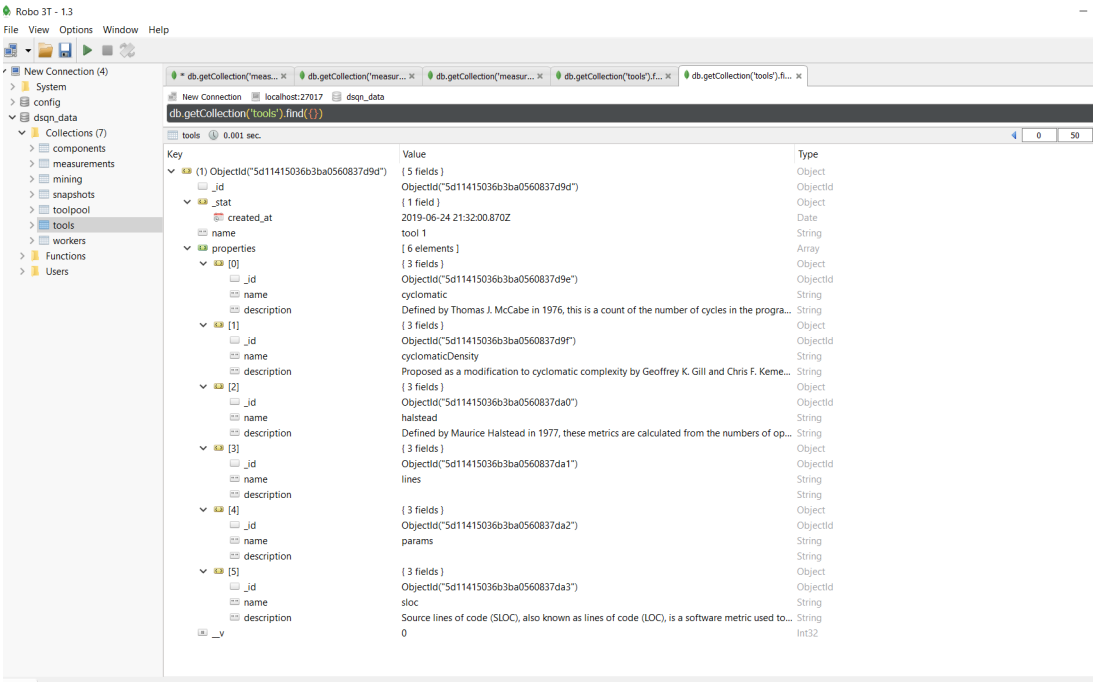

Fig. 5.10. Tool Model Schema

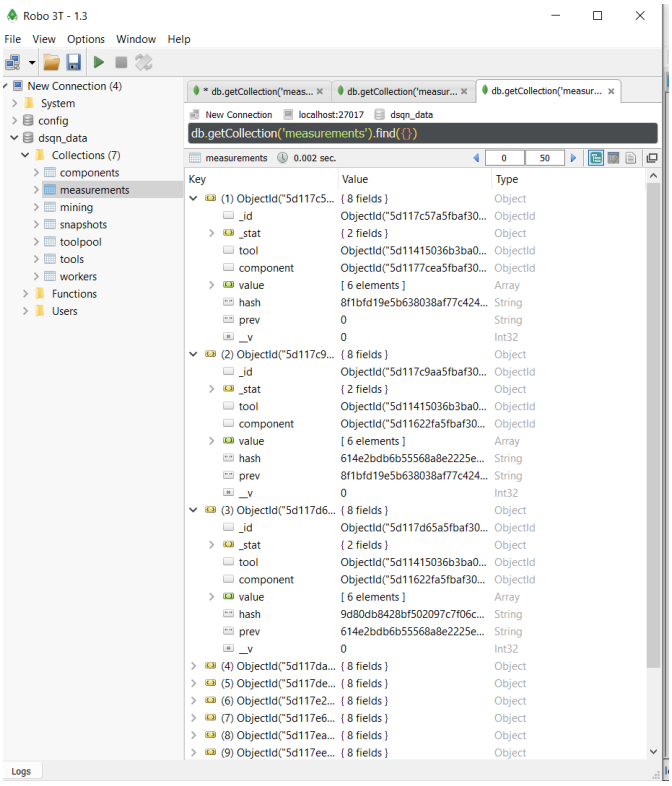

Fig. 5.11. Measurement Model Schema

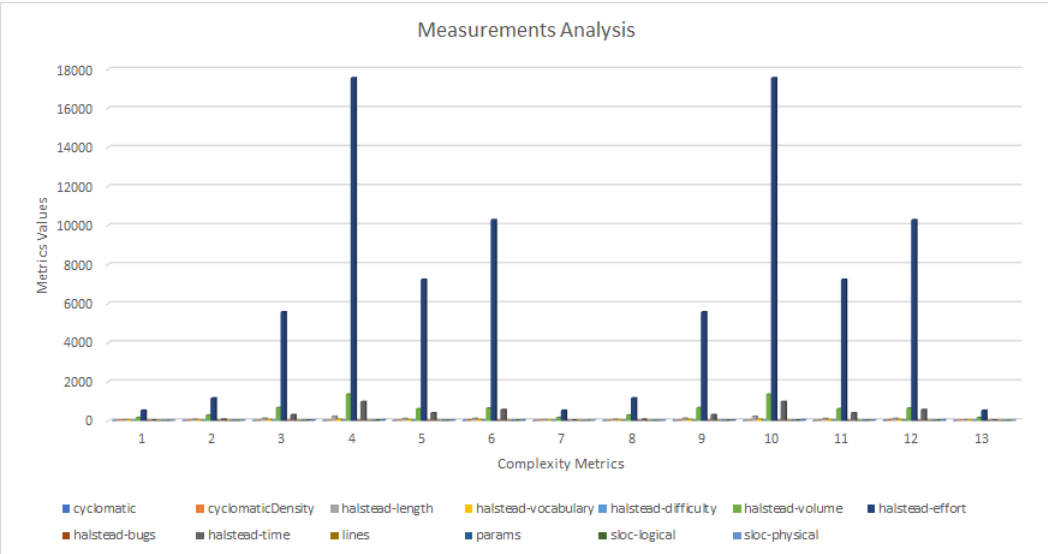

Fig. 5.12. Measurement Analysis

For experimental purpose, we fetched the node-lessons repository [33] and applied the escomplex tool on it [32]. For more detailed analysis, we have considered file *"app.js"* from initial 13 commits and obtained metrics by applying the escomplex tool as shown in graph 5.12. The graph shows the variations in the metrics after each commit. These variations can used to analyse the software advancements. These metrics can then be mined into the block after validation and a measurement chain for same can be obtained. This measurement chain can thus assist in understanding the software development process and the software itself so that its quality can also be improved.

5.4 Discussion

The prototype framework was able to successfully perform the following activities:

- Data Processing: We were able to successfully pull all the files from the GitHub repositories for each commit. Additionally, the framework was able to categorize the commit entries into specific component types: Systems, Files, and Units. Thus, the components were accurately stored into the database in their specified

data formats as described in the schema. Overall, The prototype successfully fetched the files from the GitHub repository and stored it into the database.

- Snapshot chain: We were able to create a snapshot of different components obtained from the fetched repository. The snapshot chain serves as a good historical record of the entire software development process. These records can be used to research the work history, get more insights of the project and track records after the project completion. Additionally, we can traverse the chain and obtain analysis at any stage of development by applying tools to validate that all goals are achieved as per the completeness of requirements. Furthermore, we believe this prototype can help the change management process by providing a platform for easier impact analysis, and eventually improve risk management.

- Software Assurance Aspect: We were able to successfully apply tools on different components obtained from the fetched repository. The results and analysis in section 5.3 show that the chain of measurements obtained after application of the tool serves as a good source to to track the project and measure the performance while the project is still under development. Furthermore, we believe that the Software Project Managers can use these metrics block as a resource to anticipate problems and to avoid being forced into a reactive, fix on fail approach. Overall, we were able to successfully apply multiple tools on the components and obtain the metrics, eventually leading to creation of a measurement chain.

- Prototype Testing: The blueprint framework uses the following middleware to facilitate testing: chai, chai-datetime, mocha, superagent, and supertest [23]. In our case, we have made use of Mocha for testing the prototype. For the purpose of testing, we seeded data into the prototype framework. The test was performed on different routes in the application by sending requests to paths on the prototype. The unit test included testing several routes by sending both POST and GET requests as shown in the Fig. 5.13

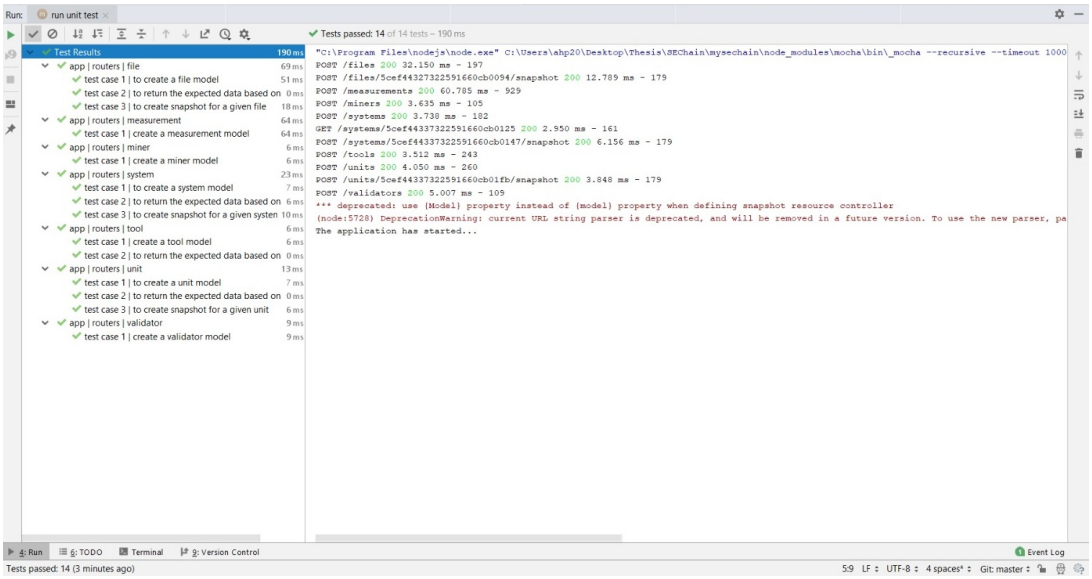

Fig. 5.13. Unit Testing : Routes

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis proposed a framework to advance the Software Quality Assurance Metrics in the area of Software Engineering and Distributed Systems. By introducing Distributed Software Quality Network we were successfully able to integrate the features of blockchain technology in the area of Software Development. The immutability feature of blockchain helped us to provide a tamper resistant data storage which we believe will eventually aid to develop trust between globally distributed team members.

Our results show that we were able to create a Snapshot level blockchain of the system, which serves as a good source for traceability testing as it provides ease for backward, forward, and bi-direction traceability. Additionally, the measurement chain served as a good source to record the software metrics while the project is under development. Furthermore, we believe that an attempt of creating a measurement chain based on the consensus theory will provide a new platform for the developers to obtain the software quality metrics at any stage of the software development process.

Based on our experience in this research, we have learned the following lessons:

- Advantage of blockchain technology:

  Integrating the concept of blockchain technology in the area of software development provides a way of documenting the historical record of the software development process while the software is under development. This historical records can be used to research the work history, get more insights of the project and track records. The immutable property of blockchain makes the record secure and trustworthy as it offers tamper resistance by using cryptography.

- Idea of implementing a distributed platform for obtaining the quality metrics:

  We believe our proposed model would not only address the challenges faced by global software development teams but would serve as a distributed platform for the developers to record their software development process and at the same time obtain quality metrics by applying tools on the different components. Thus, providing the developers with a platform to keep track of the code changes and to perform measurement analysis on any component of the project.

- Research takeaway:

  During the initial phase of this research, the scope of the project was vague. We started with basic database schema with limited attributes and project design. As the project scope became more understandable, the design scheme's changed drastically. Every change that we made had an overwork of locating the affected areas in the framework and reconciling them with the changed schema. Additionally, deciding which consensus theory works best for our use case was a challenge. Over the research phase, we analysed several consensus theories and inferred that what worked well for other blockchain usecases won't necessary suit us and that's when we decided to incorporate the product manufacture and audit process as our consensus theory. The overall research was a good learning experience for us as we went through series of failures and success.

6.2 Future Work

Based on the contribution from this thesis, we can move ahead in the following research direction. The future extension of this thesis includes implementing this proposed conceptual theory of Distributed Software Quality Network.

- Implementing and validating the proposed consensus theory.

- An interesting research direction is to implement a platform for miners to get rewarded for their efforts. Thus, the tool developers and vendors can participate in evaluating the quality metrics while getting rewarded for their services. Additionally, can serve as an exceptional platform for the independent developers to invest into obtaining software quality metrics at economical cost.

- For the purpose of experimentation, we considered using GitHub as the source for the data. The future extension could include integrating the GitHub alternatives such as Bitbucket [34], SourceForge [35], GitLab [36], Launchpad [37], codeplane [38], and Beanstalk [39] to broaden the scope of the thesis work.

REFERENCES

REFERENCES

[1] J. J. M. Trienekens, R. J. Kusters, and K. Balla, "Software process improvement, quality assurance and measurement," Sep. 2005.

[2] G. Yuan, "Study of implementation of software test management system based on web," 05 2011.

[3] M. Tihinen and J. R. Pivi Parviainen, Rob Kommeren, "Metrics and measurements in global software development," 12 2012.

[4] "What happened to software metrics?" https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5544 last Accessed: June, 2019.

[5] Z. Budimac, G. Rakic, M. Hericko, and C. Gerlec, "Towards the better software metrics tool," March 2012.

[6] D. B. Simmons, "Measuring and tracking distributed software development projects," May 2003.

[7] D. B. Simmons and N. K. Ma, "Software engineering expert system for global development," Nov 2006.

[8] C. E. L. Peixoto, J. L. N. Audy, and R. Prikladnicki, "Effort estimation in global software development projects: Preliminary results from a survey," Aug 2010.

[9] F. Hofmann, S. Wurster, E. Ron, and M. Bhmecke-Schwafert, "The immutability concept of blockchains and benefits of early standardization," Nov 2017.

[10] P. Fitsilis, V. Gerogiannis, L. Anthopoulos, and A. Kameas, "Using social network analysis for software project management," Dec 2009.

[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[12] S. Ibarra and M. Muoz, "Support tool for software quality assurance in software development," Oct 2018.

[13] R. Chatterjee and R. Chatterjee, "An overview of the emerging technology: Blockchain," 10 2017.

[14] N. Chaudhry and M. Murtaza Yousaf, "Consensus algorithms in blockchain: Comparative analysis, challenges and opportunities," 12 2018, pp. 54–63.

[15] M. Dabbagh, M. Sookhak, and N. Safa, "The evolution of blockchain: A bibliometric study," 01 2019.

[16] Y. Yuan and F.-Y. Wang, "Blockchain and cryptocurrencies: Model, techniques, and applications," 09 2018.

[17] H. Halpin and M. Piekarska, "Introduction to security and privacy on the blockchain," 04 2017.

[18] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," 10 2018.

[19] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," June 2017.

[20] S. Singh and N. Singh, "Blockchain: Future of financial and cyber security," Dec 2016.

[21] "What is hashing? [step-by-step guide-under hood of blockchain]," https://blockgeeks.com/guides/what-is-hashing/, last Accessed: June, 2019.

[22] R. S. LESLIE LAMPORT and M. PEASE, "The byzantine generals problem," July 1982.

[23] "Blueprint framework," https://blueprint.onehilltech.com/, last Accessed: June, 2019.

[24] "Mongoose - elegant mongodb object modeling for node.js," https://mongoosejs.com/, last Accessed: June, 2019.

[25] "Mongodb," https://www.mongodb.com/, last Accessed: June, 2019.

[26] "crypto-js," https://www.npmjs.com/package/crypto-js, last Accessed: June, 2019.

[27] "language-classifier," https://www.npmjs.com/package/language-classifier, last Accessed: June, 2019.

[28] "Webstrom," https://www.jetbrains.com/webstorm/, last Accessed: June, 2019.

[29] "Github," https://github.com/, last Accessed: June, 2019.

[30] "Nodegit," https://www.nodegit.org, last Accessed: June, 2019.

[31] "Robo 3t," https://robomongo.org/, last Accessed: June, 2019.

[32] "escomplex," https://www.npmjs.com/package/escomplex, last Accessed: June, 2019.

[33] "nodelessons," https://github.com/alsotang/node-lessons, last Accessed: June, 2019.

[34] "Bitbucket," https://bitbucket.org/, last Accessed: June, 2019.

[35] "Sourceforge," https://sourceforge.net/, last Accessed: June, 2019.

[36] "Gitlab," https://about.gitlab.com/, last Accessed: June, 2019.

[37] "Launchpad," https://launchpad.net/, last Accessed: June, 2019.

[38] "Codeplane," https://codeplane.com/, last Accessed: June, 2019.

[39] "Beanstalk," https://beanstalkapp.com/, last Accessed: June, 2019.